

# UE INF404 - Projet Logiciel

## Introduction

L2 Informatique

Année 2022 - 2023

# Emploi du temps et calendrier

## Emploi du temps hebdomadaire

- Cours : vendredi 9h45-11h15
- TPs : 3h dont 1h30 encadrée et 1h30 de libre-service
  - ▶ groupes INF-1 et INF-3 (à partir du 24/01)  
lundi 8h-11h15
  - ▶ groupes INF-2 et INF-4 (à partir du 27/01)  
jeudi 8h-11h15

(surveiller ADE pour les modifications éventuelles ...!)

## Calendrier du semestre

- 10 semaines de cours et TPs
- 2 semaines d'interruptions pédagogiques
- 1 semaine de partiel (le 13 mars)

→ voir également la page INF404 sur [Moodle](#) ...

# Emploi du temps et calendrier

## Emploi du temps hebdomadaire

- Cours : vendredi 9h45-11h15
- TPs : 3h dont 1h30 encadrée et 1h30 de libre-service
  - ▶ groupes INF-1 et INF-3 (à partir du 24/01)  
lundi 8h-11h15
  - ▶ groupes INF-2 et INF-4 (à partir du 27/01)  
jeudi 8h-11h15

(surveiller ADE pour les modifications éventuelles ...!)

## Calendrier du semestre

- 10 semaines de cours et TPs
- 2 semaines d'interruptions pédagogiques
- 1 semaine de partiel (le 13 mars)

→ voir également la page INF404 sur [Moodle](#) ...

# Evaluation

Trois évaluations ...

- CC1 : démonstration (orale) du projet final
- CC2 : devoir surveillé (semaine du 14 mars)
- ET : examen terminal

$$\text{Note finale} = 40\% \text{ CC1} + 20\% \text{ CC2} + 40\% \text{ ET}$$

INF404 = 3 ECTS ...

# L'UE en 2 mots ...

## Projet logiciel

- développement logiciel (INF301, INF304) en langage C  
↪ algorithmique, test et mise au point de pgms
- travail en **binômes**
- qualité du code ( $\neq$  quantité de code écrit !)

## Thème : langage et interpréteurs

- nouvelles notions théoriques ( $\sim$  INF302)  
**langage** (définition, analyse)
- compléments algorithmiques : récursivité, arbres
- démarche “systématique” pour le traitement d'un langage

## Déroulement

- un **tronc commun** sur 5 semaines
- des extensions “libres” sur 5 semaines

# L'UE en 2 mots ...

## Projet logiciel

- développement logiciel (INF301, INF304) en langage C  
↳ algorithmique, test et mise au point de pgms
- travail en **binômes**
- qualité du code ( $\neq$  quantité de code écrit !)

## Thème : langage et interpréteurs

- nouvelles notions théoriques ( $\sim$  INF302)  
**langage** (définition, analyse)
- compléments algorithmiques : récursivité, arbres
- démarche “systématique” pour le traitement d'un langage

## Déroulement

- un **tronc commun** sur 5 semaines
- des extensions “libres” sur 5 semaines

# L'UE en 2 mots ...

## Projet logiciel

- développement logiciel (INF301, INF304) en langage C  
↪ algorithmique, test et mise au point de pgms
- travail en **binômes**
- qualité du code ( $\neq$  quantité de code écrit !)

## Thème : langage et interpréteurs

- nouvelles notions théoriques ( $\sim$  INF302)  
**langage** (définition, analyse)
- compléments algorithmiques : récursivité, arbres
- démarche “systématique” pour le traitement d'un langage

## Déroulement

- un **tronc commun** sur 5 semaines
- des extensions “libres” sur 5 semaines

# L'UE en 2 mots ...

## Projet logiciel

- développement logiciel (INF301, INF304) en langage C  
↪ algorithmique, test et mise au point de pgms
- travail en **binômes**
- qualité du code ( $\neq$  quantité de code écrit !)

## Thème : langage et interpréteurs

- nouvelles notions théoriques ( $\sim$  INF302)  
**langage** (définition, analyse)
- compléments algorithmiques : récursivité, arbres
- démarche “systématique” pour le traitement d'un langage

## Déroulement

- un **tronc commun** sur 5 semaines
- des extensions “libres” sur 5 semaines



# Outline

1 Informations Générales

2 Langages et interpréteurs

3 Calculatrice

4 Le langage des expressions arithmétiques “simples” (EAS)

5 TP1 : calculatrice version 1

# Langages en informatique ? (1)

## Différents contextes d'utilisation ...

- langages de programmation  
python, ocaml, C, R, ...
- langages de commandes  
shell-scripts, Makefile, ...
- langages de “description de documents”  
DOC, PDF, PostScript, HTML, CSS, JPEG, MPEG, ...
- langages de “description de données”
- etc.

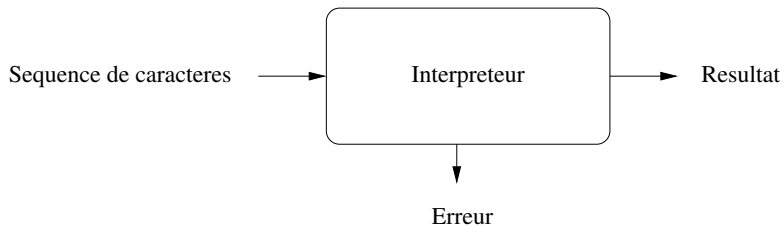
# Langages en informatique ? (2)

## Différents modes de traitement ...

En entrée : un texte  $T$  écrit dans un langage  $L$

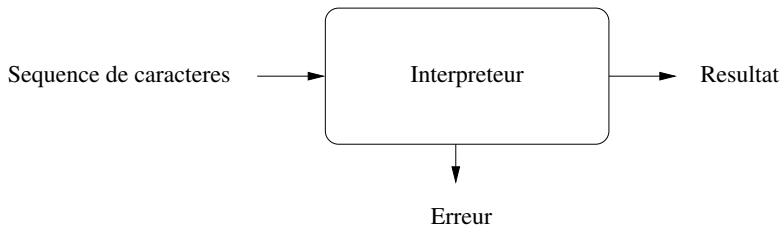
- **analyser** :  
     $\hookrightarrow$  vérifier que  $T \in L$
- **traduire (compiler)** :  
     $\hookrightarrow$  traduire  $T$  vers un langage  $L'$ , en préservant le **sens**
- **interpréter** :  
     $\hookrightarrow$  produire le **résultat**  $R$  défini par  $T$
- etc.

# Schéma général d'un interpréteur



Exemple d'interpréteurs ?

# Schéma général d'un interpréteur



**Exemple d'interpréteurs ?**

# Outline

- 1 Informations Générales
- 2 Langages et interpréteurs
- 3 Calculatrice
- 4 Le langage des expressions arithmétiques “simples” (EAS)
- 5 TP1 : calculatrice version 1

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite !) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

**démo :** la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite !) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

**démo :** la commande `bc` (basic calculator) sous Linux ...



# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite!) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

**démo :** la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite !) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

**démo :** la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite !) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

**démo :** la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite !) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

**démo :** la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite !) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

démo : la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite!) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

démo : la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : cahier des charges

**En entrée :** séquence de caractères  $S$  (clavier ou fichier)

**En sortie :**

- la **valeur** de  $S$ , si  $S$  est une **expression arithmétique correcte**
- un **message d'erreur** (explicite!) sinon ...

**Exemples :**

$5 + 2 \rightsquigarrow 7$

$5 + 2 * 3 \rightsquigarrow 11$  ( $\neq 21$ )

$(5 + 2) * 3 \rightsquigarrow 21$  ( $\neq 11$ )

$5 + * 3 \rightsquigarrow$  erreur ...

$5 \# 3 \rightsquigarrow$  erreur ...

$42 / (5 + 2 - 7) \rightsquigarrow$  erreur ...

$5/2 \rightsquigarrow 2$  (ou 2.5?)

**démo :** la commande `bc` (basic calculator) sous Linux ...

# Calculatrice : difficultés ?

*Savez-vous programmer cette calculatrice ? En combien de temps ??*

## Analyser une expression arithmétique

- lire des caractères, identifier opérateurs et opérandes
- détecter/identifier toutes les erreurs possibles

→ une **définition** du **langage des expressions arithmétiques** ?

## Evaluer une expression arithmétique

- priorité et associativité des opérateurs, parenthèses ?
- expressions non évaluables ?

## Extensions possibles :

- nouveaux opérateurs (div, mod), opérateurs unaires
- nouvelles formes pour les opérateurs (plus, moins)
- nombres décimaux, etc.



# Calculatrice : difficultés ?

*Savez-vous programmer cette calculatrice ? En combien de temps ??*

## Analyser une expression arithmétique

- lire des caractères, identifier opérateurs et opérandes
- détecter/identifier toutes les erreurs possibles

→ une **définition** du **langage des expressions arithmétiques** ?

## Evaluer une expression arithmétique

- priorité et associativité des opérateurs, parenthèses ?
- expressions non évaluables ?

## Extensions possibles :

- nouveaux opérateurs (div, mod), opérateurs unaires
- nouvelles formes pour les opérateurs (plus, moins)
- nombres décimaux, etc.

# Calculatrice : difficultés ?

*Savez-vous programmer cette calculatrice ? En combien de temps ??*

## Analyser une expression arithmétique

- lire des caractères, identifier opérateurs et opérandes
- détecter/identifier toutes les erreurs possibles

→ une **définition** du **langage des expressions arithmétiques** ?

## Evaluer une expression arithmétique

- priorité et associativité des opérateurs, parenthèses ?
- expressions non évaluables ?

## Extensions possibles :

- nouveaux opérateurs (div, mod), opérateurs unaires
- nouvelles formes pour les opérateurs (plus, moins)
- nombres décimaux, etc.

# Calculatrice : difficultés ?

*Savez-vous programmer cette calculatrice ? En combien de temps ??*

## Analyser une expression arithmétique

- lire des caractères, identifier opérateurs et opérandes
- détecter/identifier toutes les erreurs possibles

→ une **définition** du **langage des expressions arithmétiques** ?

## Evaluer une expression arithmétique

- priorité et associativité des opérateurs, parenthèses ?
- expressions non évaluables ?

## Extensions possibles :

- nouveaux opérateurs (div, mod), opérateurs unaires
- nouvelles formes pour les opérateurs (plus, moins)
- nombres décimaux, etc.

# Outline

- 1 Informations Générales
- 2 Langages et interpréteurs
- 3 Calculatrice
- 4 Le langage des expressions arithmétiques “simples” (EAS)
- 5 TP1 : calculatrice version 1

# Définir un langage ?

## Objectifs ?

- **spécifier** les entrées correctes de l'interpréteur
- classer les erreurs possibles
- permettre une programmation “systématique” (semi-automatique !)

## Définition (classique) en 4 niveaux

- 1 alphabet
- 2 lexique
- 3 syntaxe
- 4 sémantique

Dans la suite : langage des Expressions Arithmétiques Simples (EAS)

# Définir un langage ?

## Objectifs ?

- **spécifier** les entrées correctes de l'interpréteur
- classer les erreurs possibles
- permettre une programmation “systématique” (semi-automatique !)

## Définition (classique) en 4 niveaux

- ① alphabet
- ② lexique
- ③ syntaxe
- ④ sémantique

Dans la suite : langage des Expressions Arithmétiques Simples (EAS)

# Définir un langage ?

## Objectifs ?

- **spécifier** les entrées correctes de l'interpréteur
- classer les erreurs possibles
- permettre une programmation “systématique” (semi-automatique !)

## Définition (classique) en 4 niveaux

- ① alphabet
- ② lexique
- ③ syntaxe
- ④ sémantique

**Dans la suite** : langage des **Expressions Arithmétiques Simples (EAS)**

# Alphabet

*Ensemble  $V$  des caractères du langage*

## Exemples :

- en Français?

lettres de l'alphabet + lettres accentuées + ponctuation + chiffres ...

- en Anglais?

lettres de l'alphabet + ~~lettres accentuées~~ + ponctuation + chiffres ...

- en langage C?

caractères alphanumériques + ponctuation (?) + autres ...

caractère non présent dans le langage?

## Pour les EAS :

$$V = \{0, 1, \dots, 9, +, -, *, / \text{ espace, tabulation, fin-de-ligne}\}$$

On pourra ajouter :

- des lettres (pour écrire des opérateurs plus, moins, exp, log, etc.)
- le caractère ' .' (pour écrire des "nombres à virgules"), etc.

espace, tabulation, fin-de-ligne sont des **séparateurs**



# Alphabet

*Ensemble  $V$  des caractères du langage*

## Exemples :

- en Français?  
lettres de l'alphabet + lettres accentuées + ponctuation + chiffres ...
- en Anglais?  
lettres de l'alphabet + lettres accentuées + ponctuation + chiffres ...
- en langage C?  
caractères alphanumériques + ponctuation (?) + autres ...  
caractère non présent dans le langage ?

## Pour les EAS :

$$V = \{0, 1, \dots, 9, +, -, *, / \text{ espace, tabulation, fin-de-ligne}\}$$

On pourra ajouter :

- des lettres (pour écrire des opérateurs plus, moins, exp, log, etc.)
- le caractère ' .' (pour écrire des "nombres à virgules"), etc.

espace, tabulation, fin-de-ligne sont des **séparateurs**

# Alphabet

*Ensemble  $V$  des caractères du langage*

## Exemples :

- en Français?  
lettres de l'alphabet + lettres accentuées + ponctuation + chiffres ...
- en Anglais?  
lettres de l'alphabet + ~~lettres accentuées~~ + ponctuation + chiffres ...
- en langage C?  
caractères alphanumériques + ponctuation (?) + autres ...  
caractère non présent dans le langage ?

## Pour les EAS :

$$V = \{0, 1, \dots, 9, +, -, *, / \text{ espace, tabulation, fin-de-ligne}\}$$

On pourra ajouter :

- des lettres (pour écrire des opérateurs plus, moins, exp, log, etc.)
- le caractère ' .' (pour écrire des "nombres à virgules"), etc.

espace, tabulation, fin-de-ligne sont des **séparateurs**

# Alphabet

*Ensemble  $V$  des caractères du langage*

## Exemples :

- en Français?  
lettres de l'alphabet + lettres accentuées + ponctuation + chiffres ...
- en Anglais?  
lettres de l'alphabet + ~~lettres accentuées~~ + ponctuation + chiffres ...
- en langage C?  
caractères alphanumériques + ponctuation (?) + autres ...  
caractère non présent dans le langage ?

## Pour les EAS :

$$V = \{0, 1, \dots, 9, +, -, *, / \text{ espace, tabulation, fin-de-ligne}\}$$

On pourra ajouter :

- des lettres (pour écrire des opérateurs plus, moins, exp, log, etc.)
- le caractère ' .' (pour écrire des "nombres à virgules"), etc.

espace, tabulation, fin-de-ligne sont des **séparateurs**

# Alphabet

*Ensemble  $V$  des caractères du langage*

## Exemples :

- en Français?  
lettres de l'alphabet + lettres accentuées + ponctuation + chiffres ...
- en Anglais?  
lettres de l'alphabet + ~~lettres accentuées~~ + ponctuation + chiffres ...
- en langage C?  
caractères alphanumériques + ponctuation (?) + autres ...  
caractère non présent dans le langage ?

## Pour les EAS :

$$V = \{0, 1, \dots, 9, +, -, *, / \text{ espace, tabulation, fin-de-ligne}\}$$

On pourra ajouter :

- des lettres (pour écrire des opérateurs plus, moins, exp, log, etc.)
- le caractère ' .' (pour écrire des "nombres à virgules"), etc.

espace, tabulation, fin-de-ligne sont des **séparateurs**

# Lexique

Ensemble des **lexèmes** (= “mots”) du langage

## Exemples :

- en Français ?  
les mots du dictionnaire (un ensemble **fini** !)
- en langage C ?  
mots-clé (while, if), entier, opérateurs, ;, {,  
identificateurs (ensemble **infini** ?)  
exemple de lexème incorrect ?

## Pour les EAS :

Deux classes de lexèmes :

- **entiers** : séquence non vide de chiffres
- **opérateurs** : PLUS ('+'), MOINS ('-'), MULT ('\*'), DIV ('/')

# Lexique

Ensemble des **lexèmes** (= “mots”) du langage

## Exemples :

- en Français ?  
les mots du dictionnaire (un ensemble **fini** !)
- en langage C ?  
mots-clé (`while`, `if`), entier, opérateurs, `;`, `{`,  
*identificateurs* (ensemble **infini** ?)  
exemple de lexème incorrect ?

## Pour les EAS :

Deux classes de lexèmes :

- **entiers** : séquence non vide de chiffres
- **opérateurs** : PLUS (`'+'`), MOINS (`'-'`), MULT (`'*'`), DIV (`'/'`)

# Lexique

Ensemble des **lexèmes** (= “mots”) du langage

## Exemples :

- en Français ?  
les mots du dictionnaire (un ensemble **fini** !)
- en langage C ?  
mots-clé (`while`, `if`), entier, opérateurs, `;`, `{`,  
*identificateurs* (ensemble **infini** ?)  
exemple de lexème incorrect ?

## Pour les EAS :

Deux classes de lexèmes :

- **entiers** : séquence non vide de chiffres
- **opérateurs** : PLUS (`'+'`), MOINS (`'-'`), MULT (`'*'`), DIV (`'/'`)

# Lexique

Ensemble des **lexèmes** (= “mots”) du langage

## Exemples :

- en Français ?  
les mots du dictionnaire (un ensemble **fini** !)
- en langage C ?  
mots-clé (`while`, `if`), entier, opérateurs, `;`, `{`,  
*identificateurs* (ensemble **infini** ?)  
exemple de lexème incorrect ?

## Pour les EAS :

Deux classes de lexèmes :

- **entiers** : séquence non vide de chiffres
- **opérateurs** : PLUS (`'+'`), MOINS (`'-'`), MULT (`'*'`), DIV (`'/'`)



# Analyse lexicale

## Spécifier l'ensemble des lexèmes

- un lexème  $l$  = une **séquence** d'éléments de l'alphabet  $V$  ( $l \in V^*$ )
- l'ensemble des lexèmes = un sous-ensemble de  $V^*$

↪ peut être décrit par **expression régulière**

## Lexique des EAS :

- entier = chiffre.(chiffre)\*
- opérateur = '+' + '-' + '\*' + '/'

**Exo :** ajouter les “nombres réels” (ex : 3.1416, 0.5, .42, etc.)

## Analyse lexicale

- séq. de **caractères**  $\rightsquigarrow$  séq. de **lexèmes** (+ catégorie lexicale)  
[1, 2, esp, +, esp, esp 4, 5]  $\rightsquigarrow$  [entier (12), opérateur (PLUS), entier (45)]
- détecte les **erreurs lexicales**

# Analyse lexicale

## Spécifier l'ensemble des lexèmes

- un lexème  $l$  = une **séquence** d'éléments de l'alphabet  $V$  ( $l \in V^*$ )
- l'ensemble des lexèmes = un sous-ensemble de  $V^*$

↪ peut être décrit par **expression régulière**

## Lexique des EAS :

- entier = chiffre.(chiffre)\*
- opérateur = '+' + '-' + '\*' + '/'

**Exo :** ajouter les “nombres réels” (ex : 3.1416, 0.5, .42, etc.)

## Analyse lexicale

- séq. de **caractères**  $\rightsquigarrow$  séq. de **lexèmes** (+ catégorie lexicale)  
[1, 2, esp, +, esp, esp 4, 5]  $\rightsquigarrow$  [entier (12), opérateur (PLUS), entier (45)]
- détecte les **erreurs lexicales**

# Syntaxe

*Ensemble des “textes” **bien formés** du langage*

## Exemples :

- en Français?  
règles de **grammaire** (ex : sujet - verbe - complément)
- en langage C?  
 $\exists$  aussi des règles de grammaire ...  
exemple de texte C incorrect ?

## Pour les EAS :

- une EAS contient des entiers et des opérateurs
- entiers et opérateurs alternent
- un texte commence et se termine par un entier

# Syntaxe

*Ensemble des “textes” bien formés du langage*

## Exemples :

- en Français?  
règles de **grammaire** (ex : sujet - verbe - complément)
- en langage C?  
 $\exists$  aussi des règles de grammaire ...  
exemple de texte C incorrect ?

## Pour les EAS :

- une EAS contient des entiers et des opérateurs
- entiers et opérateurs alternent
- un texte commence et se termine par un entier

# Analyse syntaxique

## Spécifier les règles de syntaxe

un texte bien formé est une séquence de lexèmes

↪ peut être décrit par **expression régulière** sur les lexèmes ...

Syntaxe des EAS :

$\text{entier} . (\text{opérateur} . \text{entier})^*$

**Exo** : ajouter le “moins unaire” (ex :  $25 - -2$ ,  $-3 + 1$ ,  $- - - - 5$ , etc.)

## Analyse syntaxique

- séq. de **lexème**  $\rightsquigarrow$  Ok/Erreur  
entier (12), opérateur (PLUS), entier (45)  $\rightsquigarrow$  **Ok**  
entier (12), opérateur (PLUS), opérateur (MOINS), entier (45)  $\rightsquigarrow$  **Erreur**
- détecte les **erreurs syntaxiques**

# Analyse syntaxique

## Spécifier les règles de syntaxe

un texte bien formé est une séquence de lexèmes

↪ peut être décrit par **expression régulière** sur les lexèmes ...

**Syntaxe des EAS :**

$\text{entier} \cdot (\text{opérateur} \cdot \text{entier})^*$

**Exo :** ajouter le “moins unaire” (ex :  $25 - -2$ ,  $-3 + 1$ ,  $- - - - 5$ , etc.)

## Analyse syntaxique

- séq. de **lexème**  $\rightsquigarrow$  Ok/Erreur  
entier (12), opérateur (PLUS), entier (45)  $\rightsquigarrow$  **Ok**  
entier (12), opérateur (PLUS), opérateur (MOINS), entier (45)  $\rightsquigarrow$  **Erreur**
- détecte les **erreurs syntaxiques**

# Analyse syntaxique

## Spécifier les règles de syntaxe

un texte bien formé est une séquence de lexèmes

↪ peut être décrit par **expression régulière** sur les lexèmes ...

**Syntaxe des EAS :**

$\text{entier} \cdot (\text{opérateur} \cdot \text{entier})^*$

**Exo :** ajouter le “moins unaire” (ex :  $25 - -2$ ,  $-3 + 1$ ,  $- - - - 5$ , etc.)

## Analyse syntaxique

- séq. de **lexème**  $\rightsquigarrow$  Ok/Erreur  
entier (12), opérateur (PLUS), entier (45)  $\rightsquigarrow$  **Ok**  
entier (12), opérateur (PLUS), opérateur (MOINS), entier (45)  $\rightsquigarrow$  **Erreur**
- détecte les **erreurs syntaxiques**

# Sémantique

*Ensemble des “textes” (bien formés) du langage qui ont **un sens***

## Exemples :

- en Français ?

*la soupe mange le chien ...*

- en langage C ?

- ▶ pas de variables non déclarées
- ▶ règles de typage
- ▶ pas d'accès mémoire incorrects  
(ex : hors des tableaux, pointeurs NULL), etc.

## Sémantique des EAS :

Pas de division par 0 ...



# Sémantique

*Ensemble des “textes” (bien formés) du langage qui ont **un sens***

## Exemples :

- en Français?  
*la soupe mange le chien ...*
- en langage C?
  - ▶ pas de variables non déclarées
  - ▶ règles de typage
  - ▶ pas d'accès mémoire incorrects  
(ex : hors des tableaux, pointeurs NULL), etc.

## Sémantique des EAS :

Pas de division par 0 ...

# Sémantique

*Ensemble des “textes” (bien formés) du langage qui ont **un sens***

## Exemples :

- en Français ?  
*la soupe mange le chien ...*
- en langage C ?
  - ▶ pas de variables non déclarées
  - ▶ règles de typage
  - ▶ pas d'accès mémoire incorrects  
(ex : hors des tableaux, pointeurs NULL), etc.

## Sémantique des EAS :

Pas de division par 0 ...

# Sémantique

*Ensemble des “textes” (bien formés) du langage qui ont **un sens***

## Exemples :

- en Français ?  
*la soupe mange le chien ...*
- en langage C ?
  - ▶ pas de variables non déclarées
  - ▶ règles de typage
  - ▶ pas d'accès mémoire incorrects  
(ex : hors des tableaux, pointeurs NULL), etc.

## Sémantique des EAS :

Pas de division par 0 ...

# Semaine prochaine

- structure du programme “calculatrice” (pour les EAS !)
- Détails de :
  - ▶ analyse lexicale
  - ▶ analyse syntaxique
  - ▶ évaluation (= calcul de la valeur de l'expression ...)

# Outline

- 1 Informations Générales
- 2 Langages et interpréteurs
- 3 Calculatrice
- 4 Le langage des expressions arithmétiques “simples” (EAS)
- 5 TP1 : calculatrice version 1

# Cahier des charges de la calculatrice V1

Ecrire un interpréteur d'expressions arithmétiques

(~ “calculatrice en ligne”, comme la commande `bc` sous Linux)

Version initiale = “Expressions Arithmétiques Simples” (EAS)

- les opérandes sont des entiers
- opérateurs arithmétiques usuels (+, -, \*, /)
- pas de priorités (évaluation de gauche à droite)

Exemples :

$25 + 2$	$\rightsquigarrow$	27
$25 - 4 * 2$	$\rightsquigarrow$	42
25	$\rightsquigarrow$	25
$25 + *2$	$\rightsquigarrow$	erreur !
-25	$\rightsquigarrow$	erreur !
$25 \# 2$	$\rightsquigarrow$	erreur !
$25/0$	$\rightsquigarrow$	erreur !

# Cahier des charges de la calculatrice V1

Ecrire un interpréteur d'expressions arithmétiques

(~ “calculatrice en ligne”, comme la commande `bc` sous Linux)

Version initiale = “Expressions Arithmétiques Simples” (EAS)

- les opérandes sont des entiers
- opérateurs arithmétiques usuels (+, -, \*, /)
- pas de priorités (évaluation de gauche à droite)

Exemples :

$25 + 2$	$\rightsquigarrow$	27
$25 - 4 * 2$	$\rightsquigarrow$	42
25	$\rightsquigarrow$	25
$25 + *2$	$\rightsquigarrow$	erreur !
-25	$\rightsquigarrow$	erreur !
$25 \# 2$	$\rightsquigarrow$	erreur !
$25/0$	$\rightsquigarrow$	erreur !

# Cahier des charges de la calculatrice V1

Ecrire un interpréteur d'expressions arithmétiques

(~ “calculatrice en ligne”, comme la commande `bc` sous Linux)

Version initiale = “Expressions Arithmétiques Simples” (EAS)

- les opérandes sont des entiers
- opérateurs arithmétiques usuels (+, -, \*, /)
- pas de priorités (évaluation de gauche à droite)

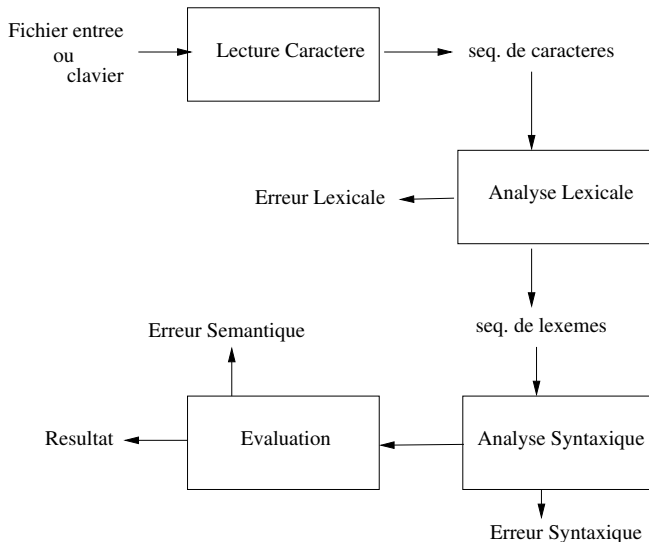
Exemples :

$25 + 2$	$\rightsquigarrow$	27
$25 - 4 * 2$	$\rightsquigarrow$	42
25	$\rightsquigarrow$	25
$25 + *2$	$\rightsquigarrow$	erreur !
-25	$\rightsquigarrow$	erreur !
$25 \# 2$	$\rightsquigarrow$	erreur !
$25/0$	$\rightsquigarrow$	erreur !



# Structure de la calculatrice

Quatre composants/modules principaux ...



# Lecture des caractères

## *Accès à une séquence de caractères*

**En entrée** : un nom de fichier (ou la chaîne vide si lecture au clavier)

**En sortie** : accès séquentiel aux caractères du fichier / msg d'erreur ...

### Primitives :

`demarrer_car`, `avancer_car`, `caractere_courant`,  
`fin_de_sequence_car`, `arreter_car`

### Implémentation :

- utilisation des primitives de `stdio.h` ...
- module fourni, à compléter/modifier le cas échéant ...

# Lecture des caractères

## *Accès à une séquence de caractères*

**En entrée** : un nom de fichier (ou la chaîne vide si lecture au clavier)

**En sortie** : accès séquentiel aux caractères du fichier / msg d'erreur ...

### Primitives :

`demarrer_car`, `avancer_car`, `caractere_courant`,  
`fin_de_sequence_car`, `arreter_car`

### Implémentation :

- utilisation des primitives de `stdio.h` ...
- module fourni, à compléter/modifier le cas échéant ...

# Lecture des caractères

## *Accès à une séquence de caractères*

**En entrée** : un nom de fichier (ou la chaîne vide si lecture au clavier)

**En sortie** : accès séquentiel aux caractères du fichier / msg d'erreur ...

### Primitives :

`demarrer_car`, `avancer_car`, `caractere_courant`,  
`fin_de_sequence_car`, `arreter_car`

### Implémentation :

- utilisation des primitives de `stdio.h` ...
- module fourni, à compléter/modifier le cas échéant ...

# Analyse lexicale

## Accès à une *séquence de lexèmes*

**En entrée** : une séquence de caractères

**En sortie** : accès séquentiel aux lexèmes / msg d'erreur ...

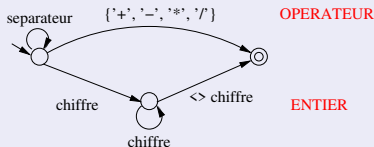
**Primitives** : définition d'un type Lexeme (public) `demarrer`, `avancer`, `lexeme_courant`, `fin_de_sequence`, `arreter`

## Implémentation :

- reconnaissance des lexèmes :

`entier = chiffre.(chiffre)*` et `operateur = '+' + '-' + '*' + '/'`

↪ **automate** :



Exo : ajouter des “nombres décimaux” ?

- module fourni, à compléter/modifier le cas échéant ...

# Analyse syntaxique ...

**En entrée** : une séquence de lexèmes

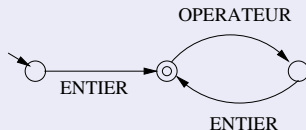
**En sortie** : valeur de l'expression arithmétique / msg d'erreur

## Primitives :

```
int analyser (char *f, int *resultat) ;  
// etat initial : f est un nom d'un fichier  
// etat final :  
    renvoie vrai si f contient une expression correcte syntaxiquement  
    dans ce cas resultat est la valeur de l'expression
```

## Implémentation :

entier.(opérateur.entier)\*  $\rightsquigarrow$  automate !



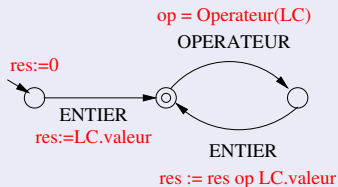
Exo : ajouter le “moins unaire” ?

## ...et évaluation !

Ni parenthèses, ni priorité d'opérateurs  $\rightsquigarrow$  évaluation de gauche à droite  
 $\Rightarrow$  évaluation possible **pendant** l'**analyse syntaxique**

## Implémentation

Etendre l'automate avec des **actions**



## Variables et fonctions auxiliaires

- `res` : valeur de l'expression (un entier)
- `LC` : lexeme courant (fourni par l'analyse lexicale)
- `Valeur(LC)` : valeur d'un lexeme ENTIER
- `Operateur(LC)` : type d'opérateur (PLUS, MOINS, etc.)

# La suite ?

## Etendre cette version

- nombres à virgules (25.2 – 7.36)
- nouveaux opérateurs (exp, modulo, plus, etc.)
- “moins unaire” ( $-25 + 12$ ,  $-- -25 + - - 12$ )

## Généralisation : priorités, donc parenthèses ...

ex :  $5 + 3 * 4 = 17$      $(5 + 3) * 4 = 32$

- nouveaux lexèmes : PARO et PARF  
→ on peut étendre l'analyse lexicale ...

## Mais :

- la syntaxe ne se décrit plus par un automate  
pas un **langage régulier** (imbrication de parenthèses)
- algo d'analyse et d'évaluation ???

⇒ **définir un nouveau formalisme ?**



# La suite ?

## Etendre cette version

- nombres à virgules ( $25.2 - 7.36$ )
- nouveaux opérateurs (exp, modulo, plus, etc.)
- “moins unaire” ( $-25 + 12$ ,  $-- -25 + -- -12$ )

## Généralisation : priorités, donc parenthèses ...

**ex :**  $5 + 3 * 4 = 17$      $(5 + 3) * 4 = 32$

- nouveaux lexèmes : PARO et PARF  
→ on peut étendre l'analyse lexicale ...

## Mais :

- la syntaxe ne se décrit plus par un automate  
pas un **langage régulier** (imbrication de parenthèses)
- algo d'analyse et d'évaluation ???

⇒ **définir un nouveau formalisme ?**